Ateneo de Davao University
School of Business and Governance
E. Jacinto St., 8016 Davao City, Philippines

Document Report on Context-Free Grammar and Right-Linear Grammar

In Partial Fulfillment for The Requirement in
Automata Theory and Formal Languages

Submitted to:
Prof. Oneil B Victoriano

Submitted by:

Dable, Nica Zoe

Delfin, Dominic Antonino

Matila, Alyssa Mhie

November 25, 2024

I. CFG Without Unit Productions

A production rule of the form A→B, where both A and B are non-terminal symbols, is referred to as a unit production. It is important to remove unit productions to optimize the grammar as they are extra symbols that increase the length of the grammar.

Procedure for Removal:

Step 1: To remove A → B, add production A → x to the grammar rule whenever B → x occurs in the grammar

Step 2: Delete A → B from the grammar

Step 3: Repeat the first and second steps until all the unit productions are removed

Example 1:
Identify and remove the unit productions from the following

S → 0X | 1Y| Z

X → 0S | 00

Y → 1| X

Z → 01

Solution:

1. Identify all unit productions: S → Z, Y → X
2. First we remove S → Z, we add S → 01 to the production because Z → 01 and take out S → Z
3. We then remove Y → X, we add Y → 0S | 00 to the production because X → 0S | 00 and take out Y → X from the production.
4. Since there's no longer a production that directs to Z, we remove Z → 01
5. The final output for the CFG is: S → 0X|1Y|01, X → 0S|00, Y → 0S|00|1

Example 2:

Identify and remove the unit productions from the following

S → Aa|B|c

B $\rightarrow$ bb|cb

A $\rightarrow$ a|bc|B

1. Identify all unit productions: S $\rightarrow$ B, A $\rightarrow$ B
2. To remove A → B, we add A $\rightarrow$ bb|cb to the production because B $\rightarrow$ bb|cb and we take out A →B
3. Next, we remove S $\rightarrow$ B, add S → bb|cb, and take out S →B
4. Since there's no production that directs to B, we remove B $\rightarrow$ bb|cb
5. The final output for the CFG is: S →Aa|c|bb|cb, A $\rightarrow$ a|bc|bb|cb

Practical Application: Robotics and Path Planning

Robots often use **CFG**s to model possible sequences of actions or commands in controlled environments. Simplified grammars ensure that action parsing and planning are efficient, especially in real-time systems like autonomous vehicles, drones, or industrial robots.

II. CFG Without Null (ε) Productions

In a CFG, a Non-Terminal Symbol 'A' is a nullable variable if there is a production A → ε or there is a derivation that starts at 'A' and leads to ε. (Like A............→ ε). The productions of type 'A → ε' are called **null productions** (also called lambda productions) (Neso Academy, 2017).

Procedure for Removal:

Step 1: To remove A → ε, look for all productions whose right side contains A

Step 2: Replace each occurrence of 'A' in each of these productions with ε

Step 3: Add the resultant productions to the Grammar

Example 1: Remove λ-productions from the following grammar

S → ABAC

A → aA | ε

B → bB | ε

C → c

Solution

1. **Eliminate A → ε:**

    Remove each A in ABAC

    S → ABAC

    S → ABC | BAC | BC

    Remove ε then substitue ε to A

    A → aA | ε

    A → aA

    A → a

    New Production, combined from old to new without null:

    S → ABAC | ABC | BAC | BC

    A → aA | a

    B → bB | ε

    C → c

2. **Eliminate B → ε**

    S → ABAC| ABC| BAC | BC

    S → AAC | AA | AC | C

    B → bB | ε

    B → bB

    B → b

    New Production, combined from previous to new without null:

    S → ABAC | ABC | BAC | BC | AAC | AA | AC | C

    A → aA | a

    B → bB | b

C → c

Example 2: Remove λ-productions from the following grammar

S → ABaC

A → BC

B → b|λ

C → D|λ

D → d

Solution:

Eliminate A → λ, B → λ, C → λ

S → ABaC

S → ABa | BaC | AaC | Aa | Ba | aC | a

A → BC

A → B | C

B → b | λ

B → b

C → D | λ

C → D

New Production, combined from old to new without null:

S → ABaC | ABa | BaC | AaC | Aa | Ba | aC | a,

A → BC|B|C,   B → b,   C → D

Practical Application: Error Handling

If a production can generate an empty string, an error recovery strategy might need to handle an extra level of complexity, as ε-productions could lead to situations where the parser unexpectedly consumes tokens or incorrectly identifies errors.

III. CFG Without Useless Productions

A variable X in a context-free grammar is called ***useless*** if it doesn't occur in any derivation of a word from that grammar (Van Glabbeek, R.,n.d.).

- Non-generating symbols: Symbols that cannot eventually produce a string of terminal symbols.
- Unreachable symbols: Symbols that cannot be reached from the start symbol via any derivation.

Steps to Eliminate Useless Productions:

Call a variable ***generating*** if it derives a string of terminals. Note that the language accepted by a context-free grammar is non-empty if and only if the start symbol is generating. Here is an algorithm to find the generating variables in a CFG:

1. Mark a variable X as "generating" if it has a production X -> w where w is a string of only terminals and/or variables previously marked "generating".
2. Repeat the step above until no further variables get marked "generating".

All variables not marked "generating" are **non-generating** (by a simple induction on the length of derivations).

Call a variable ***reachable*** if the start symbol derives a string containing that variable. Here is an algorithm to find the reachable variables in a CFG:

1. Mark the start variable as "reachable".
2. Mark a variable Y as "reachable" if there is a production X -> w where X is a variable previously marked as "reachable" and w is a string containing Y.
3. Repeat the step above until no further variables get marked "reachable".

All variables not marked "reachable" are **non-reachable** (by a simple induction on the length of derivations).

Observe that a CFG has no useless variables if and only if all its variables are reachable and generating. Therefore it is possible to eliminate useless variables from grammar as follows:

1. Find the **non-generating variables and delete them**, along with all productions involving non-generating variables.
2. Find the **non-reachable variables in the resulting grammar and delete them**, along with all productions involving non-reachable variables.

Note that step 1 does not make other variables non-generating, and step 2 does not make other variables non-reachable or non-generating. Therefore, the end result is a grammar in which all variables are reachable and generated and **hence useful.p**

Example 1: Eliminate useless symbols and productions with P consisting of:

$\mathbf{S \rightarrow A}$

$\mathbf{A \rightarrow aA \mid \lambda}$

$\mathbf{B \rightarrow bA}$

The variable B is useless and so is the production B -> bA. Although B can derive a terminal string, there is no way we can **achieve** it. So the answer is:

$\mathbf{S \rightarrow A}$

$\mathbf{A \rightarrow aA \mid \lambda}$

Example 2: Eliminate useless symbols and productions from G = (V, T, S, P), where V = {S, A, B, C} and T = {a, b}, with P consisting of:

$\mathbf{S \rightarrow aS \mid A \mid C}$

$\mathbf{A \rightarrow a}$

$\mathbf{B \rightarrow aa}$

$\mathbf{C \rightarrow aCb}$

First, we identify the set of variables that can lead to a **terminal string**. From S, A → a but this argument cannot be made for C, thus identifying it as **useless** because it's nonterminal.

$\mathbf{S \rightarrow aS \mid A}$

$\mathbf{A \rightarrow a}$

$\mathbf{B \rightarrow aa}$

Next we want to eliminate the variables that **cannot be reached** from the start variable. In our case, it shows that **B** is useless. Removing it and the affected productions and terminals, we are led to the final answer.

S → aS | A

A → a

Practical Application: Compiler Optimization

Efficient parsing: By eliminating useless productions (non-generating and unreachable symbols), the grammar becomes simpler, leading to faster and more efficient parsing of source code.

IV. CFG in Right-Linear Grammar (RLG):

A regular grammar defines a **regular language** and is represented as *G=(N, E, P, S)* where:

- *N:* A finite, non-empty set of non-terminal symbols.
- *E:* A finite set of terminal symbols (also known as the alphabet).
- *P:* A collection of production rules in one of the following forms:
  - A → *aB*
  - A → *a*
  - A → ϵ (where ϵ denotes the empty string)
    Here, A and B are non-terminals, and *a* belongs to the alphabet Σ
- *S∈N:* The start symbol.

Regular grammars fall into two categories based on the placement of non-terminal symbols in their production rules:

1. Right Linear Regular Grammar (RLG)
2. Left Linear Regular Grammar (LLG)

In a **Right Linear Regular Grammar**, non-terminal symbols always appear at the **rightmost position** in the production rules. This implies that each rule takes one of these forms:

- A→ wB
- A→ w

  Where:

  - A and B are non-terminals.

- w is a sequence of terminal symbols or the empty string ϵ

Main Features:

- Each production contains at most one non-terminal, and it must be the final symbol on the right-hand side.

The string w can be empty or consist solely of terminal symbols.

Examples :

A → a

A → aB

A → ε

A and B are non-terminals, a is terminal, and ε is an empty string

S → 00B | 11S

B → 0B | 1B | 0 | 1

S and B are non-terminals, and 0 and 1 are terminals

Definition 4.8. A CFG is called right-linear if each production body has at most one variable, and that is at the right end. That is, all productions are in the form A -> wB or A -> w, where w is a word for terminals.

Proposition 4.2. Every right-linear CFG defines a regular language.

Practical Application:

Simple Language Models: Basic NLP tasks, such as tokenization and keyword recognition, can utilize RLGs for rule-based language processing.

Syntax Checking: RLGs help create simplified models to recognize structures in sentences, although more complex grammars are needed for natural languages.

V. Converting RLG to RE

Converting a Right-Linear Grammar (RLG) into a Regular Expression (RE) involves reducing the grammar to a single expression using regular operators like union, concatenation, and the Kleene star.

We can convert an RLG into an RE using the following steps:

1. Represent each non-terminal (Ex. A, B) with an equation:
   - **Left-hand side:** The non-terminal.
   - **Right-hand side:** A combination of productions, using (+) for alternatives and concatenation for sequences.
2. Solve the equation by expressing each non-terminal using only terminal symbols and regular operators. We can do this by:
   a. Substitution

$S \rightarrow aS \mid bA$ or ($S = aS + bA$) $A \rightarrow cA \mid dA$ or ($A = cA + dA$) Substituting A into S, we have: $S = aS + b(cA+d)$ $S = aS + bcA + bd$

   b. Factorization (Removing recursive terms)

$A \rightarrow cA \mid d$ Equation Form: $A = cA + d$ Factoring Out A $A - cA = d$ $A(1-c) = d$ A = d(1-c)^-1 Since

$$(1-c)^{-1} = \sum_{n=0}^{\infty} c^n = c^*$$

Hence, A = dc*

3. Repeat 1. and 2. until no non-terminals remain. We then combine and simplify the Final Expression.
   - Once all non-terminals are eliminated, the final equation for the start symbol (SSS) is your regular expression.
   - This expression represents all strings the grammar can generate.

Example 1.

Given: S→aS | b

Solution:

Converting to equation form

S = aS + b

Isolating S

S - aS = b

Factoring S

S(1-a) = b

(1/1-a) [S(1-a)] = (1/1-a)b

S = (1-a)^-1 (b)

S = a*b

Example 2

Given:

$S \rightarrow aA \mid b$

$A \rightarrow cS \mid d$

Solution:

Converting to equation form:

$S = aA + b$

$A = cS + d$

Substituting A into S

$S = a(cS+d) + b$

$S = acS + ad + b$

$S - acS = ad + b$

$S(1-ac) = ad + b$

$S = (ad+b)(1-ac)^{-1}$

$S = (ad+b)(ac)^*$

References:

Neso Academy. (2017). Simplification of CFG (Removal of Null Productions). In YouTube. https://www.youtube.com/watch?v=mlXYQ8ug2v4

Van Glabbeek, R. (n.d.). Eliminating useless productions in context-free grammars. Stanford University. Retrieved from http://kilby.stanford.edu/~rvg/154/handouts/useless.html

BYJU's. (n.d.). Simplification of CFG notes. BYJU's. Retrieved November 24, 2024, from https://byjus.com/gate/simplification-of-cfg-notes/

Priya, B. (2021, June 16). Explain removing unit productions in context-free grammar. TutorialsPoint. Retrieved November 24, 2024, from https://www.tutorialspoint.com/explain-removing-unit-productions-in-context-free-grammar

Great Learning. (2021, July 13). Simplification of CFG: Removing null and unit productions | Compiler design tutorial. YouTube. https://www.youtube.com/watch?v=gbiPTyVKgtE

University, J., & Jaeger, B. (n.d.). *Formal Languages and Logics Lecture Notes*. https://www.ai.rug.nl/minds/uploads/LN_FLL.pdf

GeeksforGeeks. (2021, February). *Right and Left linear Regular Grammars*. GeeksforGeeks. https://www.geeksforgeeks.org/right-and-left-linear-regular-grammars/

Beckman, M. (n.d.). *Right-Linear Grammars*. Retrieved November 24, 2024, from https://courses.grainger.illinois.edu/cs421/sp2020/slides/07.2.2-right-linear-grammars.pdf

Transforming Regular Grammars to Equivalent Finite State Automata. (2024). Um.edu.mt. http://www.cs.um.edu.mt/gordon.pace/Research/Software/Relic/Transformations/RG/toFSA.html